LÖVE Workshop Übung

1 Regeln

1. Arbeite diese Aufgabenblätter selbstständig ab.
2. Auf diesen Seiten gibt es Aufgaben und Code (Quelltext/Programmcode).
3. Innerhalb einer Aufgabe (z.B. 1.1) wird der Code kontinuierlich erweitert. Wenn Du eine neue Aufgabe (z.B. 1.2) anfängst, solltest Du deine Datei leeren.
4. Funktionen (z.B. `function love.draw() ... end`) dürfen nur ein mal vorkommen.
5. Benutze die Tabulatortaste um einzurücken (links neben Q), halte Deinen Code lesbar.
6. Funktionen, Schleifen und Konditionen enden mit **`end`**. Die Zeilen vor **`end`** nennt man Körper oder Rumpf.
7. Deine Zeilen können anders durchnummeriert sein, als auf diesem Übungsblatt.

2 Malen, wie auf einer Leindwand

2.1 Dein Lieblingsrechteck

Ein Rechteck an der Position `x=100`, `y=200`. 300 Pixel breit und 150 hoch.

```
1 function love.draw()
2   love.graphics.rectangle("fill",100,200,300,150)
3 end
```

1. Zeichne das Rechteck an einer anderen Stelle.
2. Male das Rechteck „genau" in der rechten oberen Ecke. Das Fenster hat die Größe 800x600.
3. Ersetze `"fill"` mit `"line"`. Was passiert nun?
4. Male ein zweites Rechteck! Kopiere Es funktioniert nicht, eine weitere `love.draw` Funktion hinzuzufügen weil jede `love.draw` Funktion von neuem Definiert, was gezeichnet wird. Bei mehreren Definitionen zählt nur die letzte.
5. Mache das gesamte Fenster Weiß.

2.2 Zwei Rechtecke

```
1 function love.draw()
2   love.graphics.setColor(0,255,0)
3   love.graphics.rectangle("fill",100,200,300,150)
4   love.graphics.setColor(255,255,255)
5   love.graphics.rectangle("fill",300,400,100,50)
6 end
```

1. Ändere die Zahlen in Zeile 2. Was passiert?
2. Diese Farben-Repräsentation mit drei Zahlen zwischen 0 und 255 wird RGB (Rot-Grün-Blau) genannt. Färbe das kleinere Rechteck blau.
3. Bewege die Rechtecke, sodass sie überlappen. Welches ist oben?
4. Vertausche die Zeilen 3 und 5. Was ändert sich?

2.3 Ein Paar Linien

```
function love.draw()
  love.graphics.line(100,0,100,200)
  love.graphics.line(0,200,100,200)
  love.graphics.rectangle("fill",100,200,300,150)
end
```

1. Bewege das Rechteck. Passe die Linien entsprechend an.
2. Wie sorgen wir dafür, dass Rechteck und Linien sich angepasst bewegen? Mit Variablen! Lese weiter.

2.4 Variablen

```
x = 100
y = 200

function love.draw()
  love.graphics.line(100,0,x,y)
  love.graphics.line(0,200,x,y)
  love.graphics.rectangle("fill",x,y,300,150)
end
```

1. Was passiert, wenn man die Zahlen für x und y vertauscht?
2. Versuche, `x` umzubenennen.
3. Ändere Zeile 2 zu `y = x`. Was bedeutet dies?
4. Ändere Zeile 2 zurück zu `y = 200`. Ändere Zeile 1 zu: `x = y`. Es wird eine Fehlermeldung erscheinen. Kannst Du den Code korrigieren?
5. Führe eine Variable für die Breite des Rechtecks ein.

3 Interaktion

3.1 Ein bewegtes Bild

```
x = 100
y = 200

function love.draw()
  love.graphics.line(100,0,x,y)
  love.graphics.line(0,200,x,y)
  love.graphics.rectangle("fill",x,y,300,150)
end

function love.mousepressed()
  x = x + 10
end
```

1. Versuche, im Spiel den Mausknopf zu drücken. Etwas sollte geschehen. Achte darauf, wie angepasst sich die Linien verhalten.

2. Lasse das Rechteck sich rückwärts bewegen.
3. Lasse das Rechteck sich nach oben bewegen.
4. Vergrößere das Reckteck mit jedem Mausknopfdruck.
5. Füge `x = 400` nach Zeile 4 hinzu. Hättest Du das neue Verhalten erwartet?[1]

3.2 Wird richtig/rechts geklickt?

[2]

```
1  a = 100
2  b = 200
3
4  function love.draw()
5    love.graphics.rectangle("fill",a,b,300,150)
6  end
7
8  function love.mousepressed(mx, my)
9    local dir = "right"
10   if mx < love.graphics.getWidth()/2 then dir = "left" end
11   if dir == "right" then
12     a = a + 10
13   end
14 end
```

1. Wann bewegt sich das Rechteck?
2. Ersetye Zeile 9 mit `if mx < 400 and a < 200 then`. Was macht das?
3. Lasse das Rechteck den Rand berühren, aber nich darüber hinaus gehen.
4. `A or B` ist wahr (`true`), wenn wenigstens eines von beiden wahr ist. Erlaube dem Rechteck den Bildschirm zu verlassen, wenn ein bestimmter (geheimer) Bereich geklickt wird.
5. Füge einen weiteren `if`-Block nach Zeile 11 hinzu. Lasse das Rechteck sich nach oben bewegen, wenn die obere Hälfte des Bildschirms gedrückt wird.
6. `if`-Blöcke können mehr:
   ```
   if Kondition then
   dies wird ausgeführt, wenn die Kondition wahr ist
   else
   dies wird ausgeführt, wenn die Kondition nicht wahr ist
   end
   ```
   Verwende dies, um das Rechteck nach unten gehen zu lassen, wenn die untere Hälfte des Bildschirms berührt wird.
7. Gebe dem User eine Möglichkeit, die Rechtecks-Positionen widerherzustellen.

Notiz: Nach Konvention ist `A and B or C` *identisch zu* `(A and B) or C` *und demnach nicht anders als* `A and (B or C)`

[1] Warum geschieht dies? Die Spiele-Engine leert das Fenster 60 mal pro Sekunde. Der Code in `function love.draw()` wird danach *jedes mal* ausgeführt, also ist x gleich 400 jedes mal wenn die Rechtecke gezeichnet werden.

[2] Auf Touchscreens (z.B. von Handys) werden Berührungen als Klicks interpretiert.

3.3 Es soll von selbst etwas tun

```
x = 100
y = 200

function love.draw()
  love.graphics.rectangle("fill",x,y,300,150)
end

function love.update()
  y = y - 1
end
```

Alles innerhalb vom `love.update`-Block wird 60 mal je Sekunde ausgeführt.[3]

1. Halte das Rechteck am oberen Rand an.
2. Oben im Code, füge ein **velocity = 1** (Geschwindigkeit = 1). Benutze **y = y - velocity** anstelle von **y = y - 1** um das Rechteck zu bewegen.
3. Sorge dafür, dass die **velocity** dauerhaft um 0.01 geringer wird. Das simuliert Erdanziehung.
4. Gebe dem Rechteck erhöhte **velocity**, wenn diese angeklickt wird
5. Halte das Rechteck am unteren Rand an.
6. Zeige die Geschwindigkeit auf dem Bildschirm mit **love.graphics.print(velocity,10,10)**
7. Setze **velocity** gleich Null, wenn das Rechteck den oberen Rand berührt.
8. Gebe der Spielerin ein Ziel. Zeige an, wenn die Spielerin das Ziel erreicht hat.
   z.B.: Ein einfaches Parken-Spiel. Zeichne eine Linie auf der Höhe 100, ändere die Farbe des Rechteckes, wenn 0.5 > velocity > -0.5 und 105 > y > 95

3.4 Schleifen (Loops)

Die **while**-Schleife (während-loop) führt das Programm in dessen *Rumpf* oder *Körper* aus, so lange die *Schleifenbedingung* oder *Laufbedingung* **y < 500** wahr ist.

```
x = 0

function love.draw()
  y = 0

  while y < 500 do
    love.graphics.rectangle("fill",x,y,300,150)
    y = y + 200
  end
end

function love.mousepressed(mx, my)
  x = x + 50
end
```

[3] Wenn der Computer überlastet ist, wird love.update seltener ausgeführt. Mit `function love.update(dt)` wird `dt` eine Variable sein, gleich den Sekunden seit dem letzten Aufruf.

1. Verschiebe Zeile 8 unter Zeile 6. Verstehst Du den Unterschied?
2. Verschiebe Zeile 4 unter Zeile 2. Der Bildschirm sollte schwarz werden. Weshalb?[4]
3. Mache die Änderungen rückgängig. Dann zeichne mehr aber kleinere Rechtecke vertikal mithilfe der while-Schleife.
4. Füge eine neue Variable `z = 0` hinzu. Füge eine neue while-Schleife hinzu. Lasse diese `z` erhöhen und Rechtecke horizontal zeichnen.
5. Bewege `z = 0` und die neue Schleife in den Körper der alten Schleife. Nun wird durch alle z-Werte hindurchgegangen, wenn y erhöht wird! Benutze dies, um ein Schachbrettmuster zu zeichnen.
6. Bewege das Schachbrett auf Mausklick.
7. Füge hinzu:
   - **a = 1** nach Zeile 1
   - **and a == 1** zu der Bedingung der while-Schleife [5]
   - **if a == 1 then a = 0 else a = 1 end** nach Zeile 13.[6]

   Ändere den Code so, dass die Rechtecke nicht gezeichnet werden, wenn das Spiel startet, sondern erst nachdem die Maus gedrückt wird.
8. Lasse die Spielerin `a` nur ändern, wenn die Maus auf der linken Seite des Bildschirms klickt.

Die bisherigen Konzepte reichen theoretisch aus. Die folgenden Konzepte erleichtern es jedoch ungemein, zu Programmieren und Code zu Lesen.

3.5 Listen

Das *a* im Code ist eine Liste von Nummern. Hier ist a[1] gleich 100, a[2] ist 200 und a[3] ist 500. 1,2 und 3 sogenannte Indexe (indices) von 100,200 und 500 in a. `x <= y` ist wahr, wenn x kleiner oder gleich y ist.

```
a = {100,200,500}

function love.draw()
  i = 1
  while i <= 3 do
    love.graphics.rectangle("fill",a[i],a[i],10,10)
    i = i + 1
  end
end
```

1. Füge eine Nummer in die Liste `a` hinzu. Nun hat `a` 4 Elemente. `a[4]` ist das vierte Element. Mache, dass die while-Schleife alle 4 Elemente zeichnet.
2. `#a` ist die Länge von `a`.

[4] `y` wird ein mal auf 0 gesetzt, dann bis 600 erhöht und nie wieder geändert. Die while-Schleife wird nicht ausgeführt weil y = 600 und 600 > 500

[5] x == y ist wahr wenn x und y den Gleichen wert haben. Im Gegensatz dazu weist `x = y` den Wert von y an x zu.

[6] Anstelle von 1 und 0 kannst Du auch `true` und `false` verwenden. Das ist lesbarer und effizienter.

3. Du kannst das k-te Element der Liste mit `a[k] =` verändern. *z.B.: a[2] = 123* Schreibe eine while-Schleife, welche alle Werte in `a` gleich 0 setzt.

4. Du kannst auch bisher undefinierte (nil) Werte setzen. Benutze eine while-Schleife um `a` 0, 10, 20, 30, 40, 50, ..., 200 sein zu lassen

5. `a[#a+1] = v` ist das gleiche, als würde man `v` zu der Liste hinzufügen. Wanimmer die Maus geklickt wird, füge die X-Koordinate zu der Liste hinzu.

6. Erstelle zwei neue leere Listen `xs = {}` und `ys = {}`. mit jedem Klick, füge die X-Koordinate zu `xs` und die Y-Koordinate zu `xy` hinzu. Zeichne Rechtecke an allen gespeicherten Klick-Positionen.